AFCRL-67-0370

RESEARCH ON INTELLIGENT QUESTION-ANSWERING SYSTEM

By

CLAUDE CORDELL GREEN BERTRAM RAPHAEL

CONTRACT AF 19(628)-5919
PROJECT NO. 4641
TASK NO. 464102
WORK UNIT NO. 46410201

Scientific Report 1

May 1967

Contract Monitor: THOMAS G. EVANS
DATA SCIENCES LABORATORY

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

Prepared for

AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
OFFICE OF AEROSPACE RESEARCH
UNITED STATES AIR FORCE
BEDFORD, MASSACHUSETTS 01730

STANFORD RESEARCH INSTITUTE

MENLO PARK, CALIFORNIA

STANFORD RESEARCH INSTITUTE
MENLO PARK CALIFORNIA

AFCRL-67-0370

RESEARCH ON INTELLIGENT QUESTION-ANSWERING SYSTEM

By

CLAUDE CORDELL GREEN   BERTRAM RAPHAEL

SRI Project 6001

CONTRACT AF 19(628)-5919
PROJECT NO. 4641
TASK NO. 464102
WORK UNIT NO. 46410201

Scientific Report 1

May 1967

Contract Monitor: THOMAS G. EVANS
DATA SCIENCES LABORATORY

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

Approved: CHARLES A. ROSEN, MANAGER
APPLIED PHYSICS LABORATORY

J. D. NOE, EXECUTIVE DIRECTOR
ENGINEERING SCIENCES AND INDUSTRIAL DEVELOPMENT

Prepared for

AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
OFFICE OF AEROSPACE RESEARCH
UNITED STATES AIR FORCE
BEDFORD, MASSACHUSETTS 01730

ABSTRACT

This report describes progress toward an "intelligent question-answering system"--a system that can accept facts, retrieve items from memory, and perform logical deductions necessary to answer questions. Two versions of such a system have beem implemented, and the authors expect these to be the first in an evolving series of question answerers.

The first system, QA1, is based upon relational information organized in a list-structured memory. The data consist of general facts about relations as well as specific facts about objects. QA1 has limited deductive ability.

QA2 is based upon formal theorem-proving techniques. Facts are represented by statements in the predicate calculus. Although the memory organization is simpler than that of QA1, the sophisticated logical abilities of QA2 result in greater question-answering power.

The report gives examples of the performance of QA1 and QA2 on typical problems that have been done by previous question-answerers, and describes plans for extending the capabilities of QA2.

ACKNOWLEDGMENTS

Computer programming for the work reported here was done in the LISP1.5 programming language on the AN-FS-Q/32 computer time-sharing system maintained by the System Development Corporation. The authors are indebted to Mr. Robert A. Yates for his significant contributions to the implementation of program QA2.

CONTENTS

CONTENTS (Concluded)

I INTRODUCTION

A. Goals

This report describes progress during the first year of a continuing project whose primary goal is the development of an intelligent question-answering system. The point of view of this report is that a question-answering system is well-defined formal system, exhibited in the form of a computer program, that has at least the following three characteristics:

(1) The ability to accept statements of fact, possibly expressed in a formal language such as a predicate calculus, and store them in its memory.

(2) The ability to search stored information efficiently and to recognize items that are relevant to a particular query.

(3) The ability to respond appropriately to a question by identifying and presenting the answer if it is present in memory, and by deducing a reasonable logical response from relevant knowledge if the complete answer is not explicitly available.

Our present research is designed to study basic problems involved in making a question-answering system intelligent, without reference to particular potential applications. If such a system can be made practical, one can think of many ultimate applications--such as a flexible command and control information service, a reference librarian's assistant, or a robot on Mars that must be able to make the best autonomous decisions based upon limited information.

B. Previous Work

Early work in the general area of question-answering systems is competently surveyed by Simmons.[1]* Additional and more recent work has

---

* References are listed at the end of the report.

been reported by several groups including Colby and Enea,[2] McCarthy,[3] Quillian,[4] Simmons,[5] Slagle,[6], Thompson,[7] Craig *et al.*,[8] and Weizenbaum.[9]

Some of the previous work, including that of Weizenbaum and Colby and Enea, has been primarily concerned with problems related to the use of natural English as an input language. McCarthy, among others, has published several papers concerned with the interesting question of how to formalize everyday knowledge. Quillian has implemented a fully cross-referenced dictionary, which might be viewed as a useful tool for (rather than an example of) a question-answering system. Simmons, Raphael,[10] Thompson, and Craig *et al.* seem to be converging toward the viewpoint that some sort of list structure representation of relational data is necessary in a question-answering system as a general semantic model. McCarthy, in his "advice taker" discussions,[11] and Slagle and others have considered ways of applying formal logical techniques to question answering. Few of the previous projects have realistically considered the problem of extracting and utilizing only the relevant data from a potentially enormous store of facts.

Our current work is based largely upon both the semantic representations used by Raphael[12] and the formal theorem-proving techniques developed by Robinson[13] and others. We expect to demonstrate the feasibility of question-answering systems that use both list-structure semantic models and formal theorem-proving techniques to store facts, extract relevant data, and deduce logical answers to questions.

C. Methodology and Emphasis

Our original plan for this project was to design, implement, experiment with, and evaluate an evolving series of versions of a question-answering system. Each version would be designed to overcome the major shortcomings of its predecessors, and in turn the process of working with each would be expected to suggest additional features to be included in its successors.

As is well known, the process of designing, implementing, and experimenting with a large computer program can be extremely tedious and time-consuming. However, with the invaluable aid of the LISP programming language in the interactive environment of the Q-32 time sharing system, we have succeeded in carrying out the plan through two versions of the system--QA1 and QA2.

After surveying the literature, we decided that a valuable feature missing from most existing programs was a data representation that could contain general facts about the relations in the logical system as well as specific facts about the objects in the real problem domain. QA1 resulted from our search for such a memory organization. A modified predicate calculus was selected to represent information from a few of the subject areas that had been used by previous question-answering systems. Statements in this formal language were placed on property lists of various key words. Facts were retrieved by searching these property lists, and logical conclusions were deduced by using modus ponens and substitution as rules of inference. The focus of our work during this period was on techniques of memory organization.

Although QA1 achieved some limited question-answering ability, its power was limited because its logical deductive system was incomplete. Therefore we transferred our attention to the problem of providing better formal proof procedures to operate upon the information available in the question-answering system.

QA2 is a question-answering system derived almost exclusively from formal theorem-proving ideas. It is based upon the J. A. Robinson technique of resolution, and uses the unit preference strategy[14] and other heuristics to improve search efficiency.

Section II of this report discusses QA1 in detail. Section III describes formal theorem-proving techniques, including resolution, and their applicability to question answering. QA2 is discussed in Sec. IV. Section V describes some plans for future systems. Further details of the two implemented systems appear in the appendices.

II QA1--A QUESTION ANSWERER BASED ON LIST-STRUCTURED MEMORY

QA1 was the first system implemented under this project. It was based largely on the SIR system of Raphael.[9,10] The major advantage of QA1 over SIR lies in the ability of QA1 to hold in its list structure memory logical statements about how various kinds of facts might interact. Thus QA1 does not require as many separate ad hoc question-answering routines as did SIR.

QA1 is the first of what we expect to be a series of successively more powerful question answerers; it was never intended to be a complete polished system. Thus sample problem domains were borrowed from previous question-answering programs, and the control language and logical deduction programs of QA1 were left in rather rough form. In this section we describe the abilities and organization of QA1. Further details and examples are given in Appendix A. Of course, many of the capabilities of QA1 are improved in QA2; these are described later in this report.

A. Abilities of the System

In each of several subject areas, QA1 is capable of accepting facts and then answering questions based on those facts. The following are categories and examples of questions that the system is capable of answering, after the given facts have been stated in an appropriate input language:

(1) Part-whole relationships

- (a) Facts: Every person has two arms, and every arm has one hand, and every hand has five fingers.
- (b) Questions answered by the system: Does every hand have five fingers? How many fingers does a person have?

(2) Set-membership

- (a) Facts: John is an instance of a person.
- (b) Questions answered: Is John a person? John is an instance of what? Who is an instance of a person?

(3) Set-inclusion

(a) Facts: Every boy is a person.

(b) Questions: Is every boy a person? If John is a boy, who is an instance of a person?

(4) Spatial-relationships

(a) Facts: The telephone is on top of the table.

(b) Question to be answered: What is on top of the table? (If any telephone can be used for communication), what is on top of the table that can be used for communication?

(5) Miscellaneous

(a) Facts: Scott is an airfield having an aspnalt runway which is 6000 feet long. Scott, Clark, and Dover are all airfields.

(b) Questions to be answered: What is Clark? Does Scott have a runway? Does Scott have a runway length of 2000 feet?

B. Data Representation

For a discussion of some of the reasons for choosing this representation, see Sec. V-A. The data consist of relational statements similar to statements of first-order predicate calculus. They are represented by list structures organized for efficient information retrieval. (See Ref. 22 for a discussion of list structures and property lists.) Binary relations are stored in memory as attribute-value pairs on the property lists of the symbols that are related. Statements of greater complexity than binary relations are also stored on these property lists under special attributes. A sample of the data-base may be represented as follows:

JOHN → TYPE (proper name), INST (John, Boy)

PERSON → TYPE (generic name), INCLUDE (Boy, Person), HAS-PART-SET (person, arm, 2)

TELEPHONE → TYPE (generic name), INST (telephone-1, telephone), USE (telephone, communication)

USE → TYPE (relation), arguments (2)

INST → TYPE (relation), COMPUTES (INST(ST X RULE(FREEVARS(U V W) PRED(IMPLIES(AND(INCLUDE V W)(INST U V))(INST U W)))))

where the word at the tail of an arrow is an elementary symbol. The list at the head of an arrow is the property list of that symbol. INST is the name of the set-membership relation and INCLUDE is the name of the set-inclusion relation.

These statements represent the information that: "JOHN" is the name of an object belonging to the set of Boys; "PERSON" is the name of a set that includes the set of Boys; every "PERSON" has as parts a set of two arms; "Telephone-1" is a member of the set of all telephones; all telephones are used for communication; "USE" is a binary relation; "INST" is a relation; and the following axiom holds: $(\forall u)(\forall v)(\forall w)[v \subset w \wedge u \in v \Rightarrow u \in w]$.

C. Query Language

To interrogate the system, the user types questions in one of two formats, depending upon whether the question is to be answered by the FINDR function or the VFIND function.

1. FINDR

The function FINDR takes as arguments the name of a relation and arguments for that relation. Each argument for a relation, called a term, may be completely specified by name, partially specified by a descriptive operator, or unspecified (a "don't-care" term). FINDR will attempt to find objects that satisfy the specifications for each term and for which the relation holds. If it succeeds, it will return the relation with the names of the appropriate objects filled in. Although FINDR has a limited degree of logical inference ability embedded in its specification-matching facility, it is primarily a memory-searching function. It also knows about and deals with the abbreviations used in the memory.

Some examples of its operation are given below. In each case the machine's output is given on the line following the input (which starts with FINDR).

FINDR(INST(TELEPHONE1 TELEPHONE))
(INST (TELEPHONE1 TELEPHONE))

Here all terms are objects specified by name. The statement is found true and FINDR returns the input unchanged.

FINDR(INST(DCE TELEPHONE))
(INST (TELEPHONE1 TELEPHONE))

"DCE" stands for a don't-care expression--i.e., any term that satisfies the above relations.

FINDR(ONTOP((ST X USE(X COMMUNICATION))TABLE1))
(ONTOP (TELEPHONE1 TABLE1))

The first argument of the relation ONTOP is partially specified by using the descriptive operator ST ("such-that"). The term (ST X USE(X COMMUNICATION)) means "the object X such that X is used for communication." Some logical inference was necessary to answer this question. The three facts,

(a) TELEPHONE1 is a telephone,
(b) Every telephone is used for communication,
(c) TELEPHONE1 is on top of TABLE1,

were used in the course of answering this question.

2. VFIND

The function VFIND represents our first attempt at a procedure for applying special axioms found in the data. The arguments of VFIND are a list of variables and a list of predicates containing those variables. The program will then attempt to find values for the variables that will satisfy all of the predicates. If FINDR fails to find the answer by searching memory, then VFIND searches memory for a relevant axiom and then tries to deduce the answer. If FINDR succeeds, VFIND returns a list of dotted pairs in which each variable is paired with the value that satisfies the predicates. The deductive procedure is described in the next section (D). Some examples of the use of VFIND are given below.

```
VFIND((X)(INST JOHN X))
    ((X . BOY))
VFIND((X)(INST X PERSON))
    ((X . JOHN))
VFIND((Y)(HASPARTSET HUMAN FINGER Y))
    ((Y . 10))
```

This last result may be interpreted precisely to mean that every member of the set of humans has as parts a set of 10 elements from the set of fingers. Each of the above questions required several step-deductions and the use of special axioms found in memory.

D. Deduction Routines

If the answer to a given question was not found in memory, the program VFIND carried out the following procedure: Let us suppose the input is

```
VFIND((X)(INST X PERSON)),
```

meaning "find some element in the set of persons." Also, suppose that no information of the form

```
(INST MIKE PERSON)
```

is in memory. FINDR will thus fail to find the answer in the first search. It will then search for an axiom and find, on the property list of INST, the rule

```
(FREEVARS (U V W) PRED (IMPLIES (AND(INCLUDE V W)) (INST U V)) (INST U W)))
```

The consequent, (INST U W) is "matched" to (INST X PERSON) and since the match succeeds, then two subproblems consisting of the two predicates in the antecedent

```
(INCLUDE V PERSON) and
(INST JOHN V)
```

are generated. Two predicates that match these requirements,

```
(INCLUDE BOY PERSON) and
(INST JOHN BOY)
```

are then found in memory. The program keeps track of variables and returns the value

((X . JOHN))

The process is recursive and at any point in a subproblem it may call for special axioms to employ. In QA1, all axioms are in the form of an implication, so no additional rules of inference are used.

E. Evaluation and Limitations

The system was fast, taking only a few seconds of real time for the most difficult questions that it was capable of answering. Exact machine times are unknown, since the Q-32 is a time-sharing system normally having 20 to 25 users. Once the program was operative, the ability to deal with new subject areas could be added in a few minutes by merely typing in the necessary relations and axioms. Thus no reprogramming would be necessary to handle new subjects and new questions, a problem that plagued Raphael's SIR project.[12] Also, new and old subjects could be interactive. The program could employ set-membership information in solving spatial-relationship problems.

However, we felt that several changes should be made to the program. The data representation and memory organization were adequate but the deduction techniques required improvement. The program handled existential quantifiers only in certain cases and recognized only two logical connectives, AND and IMPLIES. The functions VFIND and FINDR were not quite compatible, and, as a result, the rule of inference

$$(\forall x)P(x) \Rightarrow P(a)$$

could not be applied in some cases. The program had no sophisticated means of preventing loops or picking the order in which to attempt deductions. It tried a simple depth-first search of the proof tree generated by the deductive routine described in D above. As a result of these limitations, QA1 could not answer such questions as "How many hands does John have?" and "Does there exist a person who is not a boy?" The formalization of these two problems and their solutions by QA2 are given in Sec. IV-D below.

To progress further, two alternatives were:

(1) To modify the current program by correcting each deficiency one at a time and experimentally evolve more sophisticated deductive routines, perhaps similar to those of Fischer Black's question-answering program.[15]

(2) To base our new work upon relevant research in the field of automatic theorem-proving.

Like our question-answering programs, automatic theorem-proving programs must be logically complete and must contain heuristics for selecting subproblems--i.e., for searching "proof trees" efficiently. To our knowledge, however, theorem-provers have not been used in a system containing information-retrieval capabilities. It was not clear just how a mathematical theorem-prover could be used.

We selected the second alternative--adaptation of results in automatic theorem proving--because of its potential power to provide us eventually with a very general, yet conceptually simple, question-answering system. Thus work on QA1 was abandoned and we proceeded to study how theorem-proving techniques could best be utilized, and then to implement QA2.

III THEOREM-PROVING AND QUESTION-ANSWERING

One of the most important characteristics of a question-answering system is its logical deductive ability. A system that can derive and construct responses from its stored knowledge is far more interesting than a system that can only parrot back responses that are stored explicitly in its memory.

Mathematicians and philosophers have studied the nature of implication and deduction, primarily in the abstract domain of formal logic. Most of the formal logical systems that have been studied contain all the reasonable properties one might wish in "deducing" facts in a particular, informal subject domain, and most formal systems can be easily applied, with appropriate semantic models, to the particular subject domains of interest. Therefore, we decided that, instead of developing our own heuristic deductive techniques, we should try to apply the most powerful logical procedures available in the mathematics literature to our question-answering problems.

A. Introduction to Formal Theorem-Proving Techniques

Formal logic usually deals with well-defined strings of symbols called "well-formed formulas" (wff's), and with a subset of the wff's called "theorems." Each wff can be interpreted as a statement, that may be true or false, about the state of a particular semantic model. The semantic domain may consist of any individuals and relations; in the absence of specific semantic knowledge, a domain consisting of numbers and sets is frequently used as the "standard interpretation."

A model is said to *satisfy* a wff if the statement represented by the wff is true for that model. A wff that is satisfied by all possible models (from the semantic domain) is called *valid*.

The theorems of a logical system are usually intended to be the valid wff's. However, since it is not practical in general to enumerate and test all possible models, formal syntactic procedures called proof procedures must be used to establish theorems. If every theorem of a proof procedure is indeed valid, the procedure is called sound. If every valid formula can be demonstrated to be a theorem, the procedure is complete. In the desirable case that a proof procedure is both sound and complete, the theorems of the procedure coincide with the valid wff's. A decision procedure is a sound and complete proof procedure that can effectively decide whether any given wff is valid or not.

Unfortunately, a famous theorem by Gödel shows that any sufficiently rich and consistent formal system is incomplete; that is, there will always exist wff's that are valid but cannot be formally proved to be valid. This means that, for the interesting formal systems, there can be no decision procedure; we must content ourselves with sound proof procedures that can establish as theorems some, but not all, of the valid wff's.

As a practical matter, however, the incompleteness property is much less restrictive than it may at first appear. Because of the time and space constraints on practical computation, the heuristic power of a proof procedure--i.e., its ability to prove useful theorems efficiently--is more important than its ultimate effectiveness on all theorems. A decision procedure that requires enormous amounts of time or intermediate storage is undistinguishable, in practice, from an incomplete proof procedure that never terminates for some wff's.

In recent years, much work has been done on the development on proof procedures suitable for implementation on a digital computer.[16] The most effective of these seem to be those that use the Robinson resolution principle in conjunction with the Herbrand[17] approach to theorem proving, sometimes called "semantic tableau" methods.

B. The Robinson Procedure for Proof by Resolution

The basic approach of Herbrand proof procedures is to attempt to construct a model that satisfies the negation of the wff to be proved. Since every wff is either true or false for each possible model, every model must either satisfy a given wff or else satisfy its negation, and no model can simultaneously satisfy both a wff and its negation. If a wff is valid, its negation cannot be satisfiable. If the construction of a model for the negation of a wff is completed, then the wff is not a theorem. If the construction process leads to obviously contradictory model assignments, then no satisfying model is possible and the wff is proved to be valid. If the construction process proceeds interminably, the proof procedure fails. (This is why these proof procedures are not decision procedures. It is known that no decision procedure can exist for the first-order predicate calculus.)

Proof by resolution is a Herbrand type of procedure. The negation of the wff to be proved is first placed into a standard form (prenex conjunctive normal form, in which existentially quantified variables are replaced by Skolem functions of previous universally quantified variables). In this form, the wff is represented as the conjunction of a set of formulas called clauses, each of which is a disjunction of elementary formulas called literals. Then new clauses are deduced from the starting clauses by the inference rule of resolution, such that the original wff is satisfiable only if its descendent clauses are all satisfiable. The goal of the procedure is to deduce the empty formula, which is not satisfiable and therefore demonstrates that all its antecedents, including the starting wff, are not satisfiable.

The rule of resolution is best illustrated first in its propositional form: if $p \vee \alpha$ and $\sim p \vee \beta$ are two wff's in which p is any proposition and α and β are any wff's, one may deduce the wff $\alpha \vee \beta$.

The predicate calculus form of the resolution rule is this: Let L_1 be any atomic formula--i.e., a wff consisting of a single predicate symbol followed by an appropriate set of constant, variable, and function symbols for arguments. Let L_2 be the negation of an atomic formula

consisting of the same predicate symbol as L_1, but in general with different arguments. Let α and β be any wff's in the predicate calculus. Let $(\alpha)_\sigma$ be the wff obtained from α by making all substitutions specified by the substitution set σ, of formulas for free occurrences of variables in α. If there exists any set of substitutions σ_1 of variables in L_1 and L_2 that makes L_2 identical to the negation of L_1, then from the two wff's $L_1 \vee \alpha$ and $L_2 \vee \beta$ we may deduce the "resolvent" $(\alpha \vee \beta)_{\sigma_1}$.

<u>Example</u>:

$$P(x,f(y)) \vee Q(x) \vee R(f(a),y)$$

and

$$\sim P(f(f(a)),z) \vee R(z,w)$$

imply, by resolution,

$$Q(f(f(a))) \vee R(f(a),y) \vee R(f(y),w)$$

where the substitution set is $\sigma = \{f(f(a))$ for x, $f(y)$ for $z\}$.

The main theorem of resolution states that if a resolvent is not satisfiable then neither of its antecedents are satisfiable, and that the empty formula is not satisfiable.

The resolution rule tells us how to derive a new clause from a specified pair of clauses containing a specified literal, but does not tell us how to choose which clauses to resolve. A mechanical attempt to resolve all possible pairs of clauses generally results in the generation of an unmanageably large number of irrelevant clauses. Therefore, various heuristic search principles are being developed to guide and control the selection of clauses for resolution. Among the most important of these are the set of support,[14] unit preference, level bound, and subsumption strategies.[13]

The statement of a theorem to be proved usually consists of a set of premises (<u>axioms</u>) and a conclusion. The <u>set of support strategy</u> consists of designating the conclusion, and perhaps a small number of

the most relevant axioms, as "having the T-support property"--i.e., lying in the set of support for the theorem. Thereafter only those pairs of clauses containing at least one member with T-support are considered for resolution, and every resolvant is automatically attributed the T-support property. This strategy is aimed at avoiding the deduction of consequences for some of the premises that are independent of (and irrelevant to) the particular conclusion desired.

The unit preference strategy essentially orders the clauses to be resolved by their length--i.e., by the number of literals they contain. Contradictions become apparent only when two unit (one-literal) clauses resolve together to produce the empty clause. Therefore, one might hope to discover a contradiction in the least time by working first with the shortest clauses.

Occasionally the unit preference strategy may cause one to generate and resolve lengthy, perhaps endless, sequences of unit clauses, to the neglect of longer but perhaps more fruitful clauses. This difficulty can be overcome by placing a bound on computation that will determine when the unit preference strategy should be abandoned in favor of a broader search. One such bound sets a maximum on the number of levels--i.e., intermediate steps, between a deduced clause and the original theorem.

In the course of resolution proof, several clauses may be introduced that carry equivalent information and therefore lead to distracting, extraneous steps. In particular, if C is any clause , and if $C_0 = (C)_\sigma$ is obtainable as an instance of C by some substitution set σ, and if clause $D = C_0 \vee \alpha$ where α is any formula, then C subsumes D in the sense that the set of clauses $\{C,D\}$ is satisfiable if and only if C alone is satisfiable. Therefore, we should delete from our proof any clause that is subsumed by another clause in the proof.

The proof procedure implemented as part of QA2 is a resolution procedure using some form of each of the above search strategies.

C. Utilizing a Theorem-Prover

How may this theorem-proving procedure be used in a question-answering system? We plan to use it in several ways, including the following three:

(1) Answer true-false questions. To find out if a given input sentence is true or false the theorem-prover will attempt first to prove that the sentence is true. If, after a certain expenditure of effort, no proof is found, the theorem-prover could then attempt to prove the sentence false.

(2) Find an object satisfying certain conditions. A question may be stated as, "Find x such that P(x) is true," where P(x) is some specified predicate. This problem may be posed to a theorem prover as the statement $(\exists x)(Px)$. If this statement is proved, then the answer to the question is the term that is substituted for x during the course of the proof. This term may be a variable (signifying that P(x) is true for all x), a constant, or a function. If the clause representing P(x) is used several times in the proof, then the answer is the disjunction of the several terms substituted for x. This answer may then be represented internally as a clause, e.g., $\{P(a) \vee P(b) \vee P(x) \vee P(F(y))\}$ so that the theorem prover may then simplify it by removing unnecessary disjuncts.

(3) Give the steps necessary to reach some goal. If permissible actions are suitably axiomatized, the steps in a proof that the goal is achievable correspond to the sequence of actions that must be taken to achieve it. Examples of such axiomatizations and proofs are given by McCarthy.[3] Since these axiomatizations usually involve second-order predicate calculus (e.g., statements about all predicates that can cause certain situations to occur), the Robinson procedure is not directly applicable. QA2 cannot handle this type of problem, although we expect to extend it so that it will have this capability.

Section IV-D contains annotated examples of program QA2 solving several typical problems.

IV QA2--AN EXTENDED THEOREM-PROVER AS A QUESTION-ANSWERING SYSTEM

This section contains a general description of QA2, and examples of its operation. Detailed definition of some of the theorem-prover's algorithms appear in Appendix B of this report. Listings of the complete QA2 programs, written in LISP, are available upon request.

A. General Organization

In a system having a large number of statements or facts in its memory, a key problem is that of which statement to use next in solving a problem (or proving a theorem). This problem is sometimes stated as the problem of finding which statements are "relevant" to the problem at hand.

A simple and logically complete solution to this problem is given by an extension of the set-of-support strategy; for this approach every "fact" is stored in the form of a clause suitable for use by the theorem prover.

(1) First, give the theorem-prover only the clauses representing the negation of the sentence to be proved. All clauses representing this negated sentence are given T-support. (Note that a theorem of the predicated calculus--e.g., $(\forall x)[P(x) \vee \sim P(x)]$--may be provable without reference to facts in memory.)

(2) If no proof is found, the theorem-prover then addresses memory for a limited number of additional clauses that will resolve with clauses in the theorem-prover having T-support. (Suitable memory organization and use of the subsumption test can be used to increase the efficiency of the search.)

(3) If no proof is found with the new clauses then return to Step 2.

As in other theorem-proving programs, heuristics such as a bound on level or computing time must be used to insure practical run times for the program.

This simple measure of "relevance" of one clause to another is whether or not the clauses will resolve. Note that this process is complete in the sense that if a proof exists (within the limitation on time and space) it will be found. The process is efficient in the sense that some clauses that cannot lead to a proof are never used. QA2, the program described below, contains an implementation of the above algorithm. Although this technique has the advantage of relative efficiency over giving all the clauses in memory to the theorem-prover, several improvements to this technique will be necessary before we produce a truly practical system.

B. QA2--Control Language

The question-answering program for QA2 consists of a collection of functions or subprograms which perform the various tasks necessary for such a system. At the top level an executive program EXEC allows for user-machine interaction by accepting input in its "command language," calling upon the appropriate function to perform the desired operation, and responding to the teletype user. At present, the language accepts three types of input: statements, questions, and commands.

1. Statements

A statement is entered in the following format:

S expression

where the letter S signifies that the following "expression" is to be added to the system's data base.

The expression is a predicate calculus statement such as

(IN JOHN BOY) or

((FA (X Y Z) (IF (AND (IN X Y) (INCLUDE Y Z)) (IN X Z)))

The first states that John is a boy, or more precisely, that John is an element of the set named Boy.

The second is equivalent to the predicate calculus statement:

$$(\forall x)\ (\forall y)\ (\forall z)\ [\ x \in y \wedge y \subset z \Rightarrow x \in z\ ]$$

When a statement is encountered, it is transformed into prenex conjunctive normal form and then filed in the memory of the question-answering system.

2. Questions

A question is entered in a similar fashion:

Q question

where Q signifies that the predicate calculus expression that follows is to be treated as a question to the system. Here, the negation of the question is put into conjunctive normal form and passed on to a subexecutive program EXEC1 which attempts to answer the question based on the current information in the data base. (Part D of this section shows how various questions may be posed as predicate calculus expressions.)

3. Commands

A series of additional commands have been implemented which allow the user to interrogate and alter the system:

(a) UNWIND

After a question has been successfully answered, the UNWIND command will print the proof of the answer given to the question.

(b) CONTINUE

If the system was unsuccessful in answering a question, the CONTINUE command will cause the system to continue searching for proof with the level bound raised.

(c) LIST

The command LIST PR where PR is a predicate symbol will list all of the statements in the data base that contain the symbol PR.

(d) FORGET

The command FORGET PR S will delete certain statements that contain the predicate letter PR according to the format of S--e.g., if S is an integer n, the n^{th} statement will be deleted.

(e) FILE

FILE F asks the theorem prover to operate on a prepared list F of clauses.

C. The Theorem Prover

The theorem prover FNI accepts as input a list of clauses CLAUSELIST and a level bound MAXLEV. Its goal is to determine, if possible, that the set of clauses is unsatisfiable, or equivalently, to derive the null clause (containing no literals) which is the result of two contradictory clauses.

The algorithm used is the unit preference strategy with T-support. In addition, it traces the values assigned to variables that were originally bound by existential quantifiers. Thus if the theorem prover completes a proof of a statement of the form $(\exists x)\ P(x)$, the question answerer can exhibit the x that satisfies P. This is extremely useful, as the examples in part D below will show.

The operation of the theorem prover starts by ordering the clauses on CLAUSELIST by the number of literals in each clause. The program successively attempts to produce resolvents from the clauses in CLAUSELIST, producing first those resolvents of shortest length. To avoid redundant computation as much as possible, resolvents of two clauses C1 and C2 are produced only if the following criteria are satisfied:

(1) Either C1 or C2 (or both) must have T-support.

(2) The level ℓ of any resolvent of C1 and C2 plus the length of the resolvent must not be greater than the level bound MAXLEV. (This is a modification of the usual level bound strategy.)

(3) Neither C1 nor C2 has been subsumed by any other clause in the proof.

Furthermore, if a resolvent R of C1 and C2 is produced, it is added to CLAUSELIST only if R is not a tautology--i.e., does not contain complementary literals--and if R is not subsumed by any clause already on the list.

D. Examples of QA2 in Operation

A sample dialogue with QA2 is given below. The input and output from the computer are printed in all capital letters. After some of the exchanges, we have added an explanation.

```
S    (IN JOHN BOY)
     OK
```

The statement (indicated by "S") that John is contained in the set of boys is accepted and the response is "OK."

```
Q    (IN JOHN BOY)
     YES
```

The question (indicated by "Q"), "Is John in the set of boys?" is answered "Yes." This is an example of a simple yes or "no proof found" answer.

```
Q    (EX (X) (IN JOHN X))
     YES WHEN X = BOY
```

Does there exist an x such that John is in the set x? Note that the program reports what assignment is made to x to complete its proof.

```
S    (FA (X) (IF (IN X BOY) (IN X PERSON)))
     OK
```

This says that every boy is a person, or $(\forall x)[x \epsilon BOY \Rightarrow x \epsilon PERSON]$

```
Q    (EX (X) (IN X PERSON))
     YES WHEN X = JOHN
```

Does there exist a member of the set of humans? The theorem prover must have used two statements: John is a boy, and every boy is a person.

```
UNWIND

 SUMMARY
 1   IN(JOHN,BOY)                             AXIOM
 2   -IN(X,PERSON)                            NEG OF THM
 3   -IN(X,BOY)   IN(X,PERSON)                AXIOM
 4   -IN(X,BOY)                               FROM 2,3
 (CONTRADICTION FROM CLAUSES 1 AND 4)

 (5 CLAUSES GENERATED)
```

The command 'NWIND caused the proof to be printed out. Each numbered line corresponds to one clause. A clause may come from three sources:

AXIOM	-	retrieved from memory
NEG OF THM	-	the negation of the question
FROM N,M	-	the result of resolving together clauses N and M.

The number of clauses generated represents the size of the proof tree upon generating the empty clause; this is a measure of the amount of effort involved in completing the proof.

```
S    (FA (X) (IF  (IN X PERSON)  (IN X HUMAN)))
     OK
```

It unquestioningly believes that all persons are human.

```
Q    (EX (X) (IN X HUMAN))
     YES WHEN X = JOHN

S    (FA (X) (IF (IN X HUMAN) (HP X ARM 2)))
     OK

Q    (HP JOHN ARM 2)
     YES
```

(HP JOHN ARM 2) means that John Has-as-Parts two elements of the set of all arms.

```
S    (FA (Y) (IF (IN Y ARM) (HP Y HAND 1)))
     OK

Q    (EX (X) (HP JOHN HAND X))
     NO PROOF FOUND
```

The crucial axiom, given next, was missing.

```
S    (FA (X Y Z M N) (IF (AND (HP X Y M)
        (FA (U) (IF (IN U Y) (HP U Z N)))) (HP X Z (TIMES M N))))
     OK

Q    (EX (N) (HP JOHN HAND N))
     YES WHEN N = TIMES (2,1)
```

TIMES (2,1) represents the product of 2 and 1 (=2).

UNWIND

SUMMARY

```
1    IN(JOHN,BOY)                                                  AXIOM
2    -HP(JOHN,HAND,N)                                              NEG OF THM
3    IN(SK8(N,M,Z,Y,X),Y)    - HP(X,Y,M)
      HP(X,Z,TIMES(M,N))                                           AXIOM
4    -HP(JOHN,Y,M)   IN(SK8(N,M,HAND,Y,JOHN),Y)                    FROM 2,3
5    -IN(Y,ARM)   HP(Y,HAND,1)                                     AXIOM
6    -HP(JOHN,ARM,M)   HP(SK8(N,M,HAND,ARM,JOHN),HAND,1)           FROM 4,5
7    -HP(SK8(N,M,Z,Y,X),Z,N)   -HP(X,Y,M)
      HP(X,Z,TIMES(M,N))                                           AXIOM
8    -HP(JOHN,Y,M)   -HP(SK8(N,M,HAND,Y,JOHN),HAND,N)              FROM 2,7
9    -HP(JOHN,ARM,M)                                               FROM 6,8
10    -IN(X,HUMAN)   HP(X,ARM,2)                                   AXIOM
11    -IN(JOHN,HUMAN)                                              FROM 9,10
12    -IN(X,PERSON)   IN(X,HUMAN)                                  AXIOM
13    -IN(JOHN,PERSON)                                             FROM 11,12
14    -IN(X,BOY)   IN(X,PERSON)                                    AXIOM
15    -IN(JOHN,BOY)                                                FROM 13,14
(CONTRADICTION FROM CLAUSES 1 AND 15)
 27 CLAUSES GENERATED)
```

This required a 8-step proof. SK8 is the name generated by the program for a Skolem function used to eliminate an existential quantifier.

```
S     (OR (AT JOHN HOME) (AT JOHN SRI) (AT JOHN AIRPORT))
      OK
Q     (EX (X) (AT JOHN X))
      YES   WHEN X = SRI
         OR   WHEN X = AIRPORT
         OR   WHEN X = HOME
```

Note that the output may be a logical "OR" of several possibilities.

```
S (FA (X) (EQUALS X X))
 OK

S (FA (X Y) (IF (EQUALS X Y) (EQUALS Y X)))
 OK

S (FA (X Y Z) (IF (AND (EQUALS X Y) (EQUALS Y Z)) (EQUALS X Z)))
 OK

S (FA (X Y Z) (IF (AND (AT X Y) (AT X Z)) (EQUALS Y Z)))
 OK
```

```
S (NOT (EQUALS SRI AIRPORT))
 OK

S (NOT (EQUALS AIRPORT HOME))
 OK

Q (EX (X) (IF (NOT (AT JOHN AIRPORT)) (AT JOHN X)))
     YES    WHEN   X = HOME

       OR   WHEN   X = SRI

S (IF (AT JOHN AIRPORT) (WITH JOHN BILL))
 OK

S (FA (X Y Z) (IF (AND (AT X Y) (WITH Z X)) (AT Z Y)))
 OK

Q (EX (X) (IF (AT JOHN AIRPORT) (AT BILL X)))
 NO PROOF FOUND

S (FA (X Y) (IF (WITH X Y) (WITH Y X)))
 OK

Q (EX (X) (IF (AT JOHN AIRPORT) (AT BILL X)))
     YES    WHEN   X = AIRPORT

Q (EX (X) (IF (NOT (WITH BILL JOHN)) (AT JOHN X)))
     YES    WHEN   X = SRI

       OR   WHEN   X = AIRPORT

       OR   WHEN   X = HOME

S (AT JOHN SRI)
 OK

Q (NOT (AT JOHN AIRPORT))
 YES

S (FA (X Y) (IFF (DISJOINT X Y) (FA (U)
    (IF (IN U X) (NOT (IN U Y))))))
 OK

Q (FA (X Y) (IF (DISJOINT X Y) (DISJOINT Y X)))
 YES

S (DISJOINT BOY GIRL)
 OK

S (IN JOHN BOY)
 OK
```

```
Q (NOT (IN JOHN GIRL))
 YES

S (IN JUDY GIRL)
 OK

S (FA (X Y Z) (IF (AND (IN X Y) (INCLUDE Y Z)) (IN X Z)))
 OK

S (INCLUDE BOY PERSON)
 OK

Q (EX (X) (IN X PERSON))
    YES   WHEN  X = JOHN

S (INCLUDE GIRL PERSON)
 OK

Q (EX (X) (AND (NOT (IN X BOY)) (IN X PERSON)))
    YES   WHEN  X = JUDY

UNWIND

 SUMMARY

 1    DISJOINT(BOY,GIRL)                                  AXIOM
 2    INCLUDE(GIRL,PERSON)                                AXIOM
 3    IN(JUDY,GIRL)                                       AXIOM
 4    IN(X,BOY)  -IN(X,PERSON)                            NEG OF THM
 5    -INCLUDE(Y,Z)  -IN(X,Y)
       IN(X,Z)                                            AXIOM
 6    IN(X,BOY)  -IN(X,Y)
       -INCLUDE(Y,PERSON)                                 FROM 4,5
 7    -INCLUDE(GIRL,PERSON)  IN(JUDY,BOY)                 FROM 3,6
 8    IN(JUDY,BOY)                                        FROM 2,7
 9    -DISJOINT(X,Y)  -IN(U,X)
       -IN(U,Y)                                           AXIOM
10    -IN(JUDY,Y)  -DISJOINT(BOY,Y)                       FROM 8,9
11    -IN(JUDY,GIRL)                                      FROM 1,10
(CONTRADICTION FROM CLAUSES 11 AND 3)

(92 CLAUSES GENERATED)
```

V CONCLUSIONS AND SPECULATIONS

A. The Problem of Data Representation

Suppose we wish to store in a computer some items of information, which we shall call "the data," for use by a question-answering system. This data may have as its source English text, logical rules, pictures, etc. The problem of representing this data may be divided into three parts:

(1) Determining the semantic content of the data. For example, we may decide that the semantics of the sentence, "John is the father of Bill," is expressed by the binary relation "is-the-father-of" applied to the objects named "John" and "Bill."

(2) Choosing a language in which to express this semantic content. For example, we may use the notation of first order logic and pick appropriate symbols--i.e.,

Father (John, Bill)

(3) Choosing a memory organization--i.e., a way to represent statements in the computer memory. In the LISP programming system, for example, statements would be stored in linked list structure, possibly using LISP atomic symbols as entry points--e.g., on the property-list of the atom "John" we could place the value "Bill" under the attribute "Father."

In expressing the semantic content of, say, a sentence of English, we are deciding what information that sentence can provide for the question-answering system. More specifically, we are restricting the set of statements that may be deduced from the representation of that sentence. Thus a criterion that should be used in specifying semantic content is: Will the system be able to correctly answer questions concerning the subject matter of that sentence?

The language mentioned in (2) above should be selected to represent, unambiguously and compactly, the semantic content of the data. This

language should also be such that we can write a set of rules (a program) to manipulate the facts expressed in this language in order to produce the desired answer.

It is not yet clear what criteria to use in selecting the memory organization or data structure for a given language. In particular, the representation of a given statement could be either one or the other of the following:

(1) In such a form that the structure of the statement itself aid in information retrieval.

(2) In such a form that stored items may be easily used in the deductive routines--e.g., as clauses for a proof procedure. In this case additional structure must be provided to aid in information retrieval.

In any case, it soon becomes difficult to study the question of memory organization apart from the question of problem-solving or deductive techniques.

Our work thus far suggests that the fastest progress may be obtained by first planning problem-solving techniques and deductive routines, then choosing a language to represent the data that is compatible with these techniques, and, finally, selecting a memory organization compatible with the problem-solving techniques and languages chosen. However, since the efficiency of memory organization, language representation, and problem-solving techniques are highly interdependent, we expect future versions of our question-answering systems will exhibit changes in all these aspects.

At our current stage of work, the greatest advances in question-answering ability seem to depend upon improved logical problem-solving ability. The choice of internal language and data structures has not yet been critical to the performance of the systems.

B. Improvements to QA2

At the time of this writing, QA2 is an operational program with the ability to accept and store facts in the form of predicate calculus statements, and to answer questions by (1) selecting relevant facts from

its memory, (2) operating a theorem-proving program on selected facts, and (3) keeping track of the bindings of variables made in the course of a proof. However, the measure of relevance for selecting facts is very rudimentary; the theorem prover is the Robinson procedure with a few basic search heuristics; and the tracking of variables is only a first step toward constructing answers to complex questions. In each of these areas, several potential modifications are apparent that would result in improved question-answering ability.

1. Selecting Relevant Facts

The present criterion of relevance is, "Is the selected fact resolvable, by the theorem prover, with any clause having T-support?" As the data base grows, more stringent relevancy criteria will have to be used. One new relevancy measure we plan to use is the number of constants and predicate symbols in common between the fact in memory and the clause with which it resolves. Of course, the efficient use of such conditions implies that facts in memory must be indexed or sorted in appropriate ways.

2. Improving the Theorem Prover

Several modifications to the theorem prover would result in significant reductions in running time with no decrease in effectiveness.

Theoretical work by Hart,[18] Robinson,[22] and Slagle,[10] indicates that many unnecessary steps are performed by a basic theorem prover such as the one we have implemented. Variations of techniques to avoid the extra steps have been called "Elimination of equivalent proofs," "Finding maximal clashes," and "Making a theorem prover singly connected." We feel that the increased efficiency resulting from implementing one of these procedures will probably justify the increased overhead necessary for bookkeeping.

Another major increase in theorem-proving efficiency would result from incorporating into the theorem prover some special knowledge about certain relations. If equality is defined by three axioms in the system to be an equivalence relation, then every proof involving equality will make many superfluous references to those axioms merely because of their

extreme generality. These extraneous proof steps would be avoided if special information about the significance of equality were built right into the theorem-proving program--e.g., if we are given that $f(b) = f(a)$, then $P(a, x, f(x))$ should resolve directly with $\sim P(a, b, f(a))$, without intermediate use of equality axioms. Similarly set-inclusion and set-membership relations occupy distinguished roles in most logical deductions and thus should be given special treatment.

3. Increasing the Range of Question Types

Currently QA2 can answer two types of questions: "Is S true?" is answered by presenting the predicate calculus sentence S to the theorem-prover; "Find an x satisfying $P(x)$" is answered by presenting the formula $(\exists x)\ P(x)$ to the theorem prover and tracking the values it assigns to x. We plan to study how the system can be extended to handle additional question types. In particular, many axiomatizations of interesting problems, such as problems whose solutions consist of a sequence of steps or actions require second-order predicate calculus. We believe we can extend the resolution technique to handle certain forms of second-order theorems.

C. Further Plans

We have discussed above two experimental question-answering systems: QA1, on which work has stopped, and QA2, which is operational but for which various improvements are now in progress. Our work on these systems has suggested several major innovations that we expect to incorporate into future question-answering systems.

1. Extensions of the Logical Notation

Various notational devices, some of which can be defined as abbreviations in conventional logical systems, could have a special significance in a question-answering system. For example, consider, in addition to the usual universal and existential quantifiers $(\forall x)$ and $(\exists x)$, the following special quantifier-type operators:

There exists a unique element	$(\exists ! x)$	
All x in the set y	$(\forall x \epsilon y)$	epsilon quantifiers
There exists an x in set y	$(\exists x \epsilon y)$	
Any object x	(ST x)	
The set of all x	(STA x)	
unique object x	(ST! x)	descriptive operator.

The descriptive operator is discussed by Kalish and Montague.[20] The "ST" (such that) operator was useful in the QA1 representation and query language.

Epsilon quantifiers can be helpful, not only in expressing facts and questions more naturally, but also in directing search procedures. Such use of quantifiers that reduce the size of the search space can be even more effective if the theorem prover has special built-in knowledge of the properties of the relevant relations.

2. Enhancement of the Man-Machine Interface

The operation of the system should be made more convenient and more transparent to the human user. The interface language should eventually be made more English-like. (The QA2 interface language is already slightly more natural than standard logical prefix notation.) In addition, the system should be made more interactive. For example, the system should be able to report why or how it gets into difficulty on a proof, and request assistance when necessary.

3. Use of New Planning and Search Heuristics

Slagle[19] reports a technique for finding clashes, and thus eliminating equivalent proofs, by making use of semantic models for the logical system. A generalization of this technique might permit the theorem prover to use facts in the question-answerer's memory as a guide to determining which steps to try next. Extensive work remains to be done on how a theorem-prover can best utilize semantic information.

Better planning heuristics could significantly reduce the time used to produce a proof (and thus answer a question). The choice and proof of appropriate lemmas as "stepping stones," along with an extended set of support strategy, could provide a sense of direction that is missing from most current theorem-proving programs.

Appendix A

OPERATION OF QA1

Appendix A

OPERATION OF QA1

1. Data File

All the data contained in memory may be shown by listing the file "Dictionary." This computer output is shown below. The first word in each list is a LISP atomic symbol and the rest of the list is the property list of that symbol.

```
LIST DICTIONARY
     LISTING OF FILE DICTIONARY

  (TEL1 TYPE PROPNAME INST (TEL1 TELEPHONE) ONTOP (TEL1 TABLE1))

  (TYPE TYPE RELATION ARGS 2 EQUIVFORM ((PL EQUIVFORM RULE (FREEVARS
       (X Y) PRED (EQUIV (PL X TYPE Y) (TYPE XY))))))

  (MIKE TYPE PROPNAME INST (MIKE PERSON))

  (TABLE1 TYPE PROPNAME INST (TABLE1 TABLE) ONTOP (TEL1 TABLE1))

  (GIRL TYPE GENPHYS INCLUDE (GIRL PERSON))

  (PLUSS TYPE FUNCTION)

  (HASPART TYPE RELATION ARGS 2 INPUT ((STF (X Y)
       TYPE (X OBJECT) TYPE (Y OBJECT))))

  (SAM TYPE PROPNAME)

  (HASPARTSET TYPE RELATION DEFINITION ((PL DEFINITION INTERMSOF
       (HASPART CARDINALITY)
        RULE (FREEVARS (X Y Z N)
         PREDICATE (EQUIV (HASPARTSET X (Y N))
          ((XISTSU Z INST (Z Y))
           (FORALL W INST (W Z))
           (AND (HASPART X W) (CARD Y N)))))))
   COMPUTES (HASPARTSET (ST X RULE (FREEVARS (X Y Z M N)
      PRED (IMPLIESS (ANDD (HASPARTSET X (L Y M))
        (HASPARTSET Y (L Z N)))
       (HASPARTSET X (L Z (EV TIMESS M N))))))))
```

```
(AIRFIELD TYPE GENPHYS INST ((XLIST SCOTTT CLARK DOVER) AIRFIELD)
  HASPART (AIRFIELD (A RUNWAY)))

(FINGER TYPE GENPHYS HASPARTSET (HAND (L FINGER 5)))

(USE TYPE RELATION ARGS 2)

(PERSON TYPE GENPHYS INCLUDE (BOY PERSON) HASPARTSET (PERSON (L ARM 2)))

(BOY TYPE GENPHYS INST (JOHN BOY) INCLUDE (BOY PERSON))

(ARM TYPE GENPHYS HASPARTSET (ARM (L HAND1)) HASPARTSET (PERSON
    (L ARM 2)))

(RUNWAY TYPE GENPHYS HASPART (AIRFIELD (A RUNWAY)))

(BOOK1 TYPE PROPNAME INST (BOOK1 BOOK)
    COLOR (BOOK1 (ST X INST (X MIXEDCOLOR( PART (X RED) PART (X BLUE))))

(TIMESS TYPE FUNCTION ARGS INDEF COMPUTESL TIMES)

(HAND TYPE GENPHYS HASPARTSET (HAND (L FINGER 5))
    HASPARTSET (ARM (L HAND 1)))

(SCOTT TYPE PROPNAME INST (SCOTT AIRFIELD)
    HASPART (SCOTT (ST X INST (X RUNWAY)
    SURFACE ASPHALT LENGTH (6000 FEET))))

(JOHN TYPE PROPNAME INST (JOHN BOY))

(INST TYPE RELATION COMPUTES (INST (ST X RULE (FREEVARS (U V W)
    PRED (IMPLIESS (ANDD (INCLUDE V W) (INST U V)) (INST U W))))))

(TELEPHONE TYPE GENPHYS INST (TEL1 TELEPHONE)
  USE (TELEPHONE COMMUNICATION) USE PAPERWEIGHT

    END OF FILE DICTIONARY
```

2. Results with QA1

With the data base given in the above "Dictionary," the following question and answers session was conducted. The first line in each group is input, followed by the system's response.

```
FINDR( ONTOP( TEL1 TABLE1,)
 (ONTOP (TEL1 TABLE1))

FINDR( ONTOP( XXA TABLE1))
 (ONTOP (TEL1 TABLE1))

FINDR(ONTOP( TEL1 XXA))
 (ONTOP (TEL1 TABLE1))

FINDR(ONTOP (TEL1 TABLE2))
 NIL

FINDR( ONTOP( TEL2 TABLE1))
 NIL

FINDR(INST ( XXA TELEPHONE))
 (INST (TEL1 TELEPHONE))

FINDR( TYPE( TEL1 XXA))
 (TYPE PROPNAME)

FINDR( INST( MIKE XXA))
 (INST (MIKE PERSON))

FINDR( INCLUDE( BOY XXA))
 (INCLUDE (BOY PERSON))

FINDR( TYPE (PLUSS XXA))
 (TYPE FUNCTION)

FINDR( TYPE( SAM XXA))
 (TYPE PROPNAME)

FINDR( DEFINITION( HASPARTSET DCE)))
 (DEFINITION ((PL DEFINITION INTERMSOF (HASPART CARDINALITY)
     RULE (FREEVARS (X Y Z N)
      PREDICATE (EQUIV (HASPARTSET X (Y N))
       ((XISTSU Z INST (Z Y))
        (FORALL W INST (W Z)) (AND (HASPART X W) (CARD Y N))))))))

FINDR( INST (CLARK AIRFIELD))
 (INST ((XLIST SCOTTT CLARK DOVER) AIRFIELD))

FINDR(HASPART (AIRFIELD (A RUNWAY)))
 (HASPART (AIRFIELD (A RUNWAY)))

FINDR(HASPARTSET (HAND (L FINGER 5)))
 (HASPARTSET (HAND (L FINGER 5)))
```

```
FINDR( HASPARTSET(HAND(L DCE 5)))
 (HASPARTSET (HAND (L FINGER 5)))

FINDR(HASPARTSET(DCE(L FINGER 5)))
 (HASPARTSET (HAND (L FINGER 5)))

FINDR(HASPARTSET(HAND(L FINGER DCE)))
 (HASPARTSET (HAND (L FINGER 5)))

FINDR( HASPARTSET( ARM(L HAND DCE)))
 (HASPARTSET (ARM (L HAND 1)))

FINDR(HASPARTSET( ARM(L FINGER DCE))
 NIL
    *Note that FINDR cannot solve this, whereas VFIND can (see below).

FINDR( HASPART (SCOTT DCE)))
 (HASPART (SCOTT (ST X INST (X RUNWAY)
     SURFACE ASPHALT LENGTH (6000 FEET))))

FINDR(HASPART (SCOTT (A RUNWAY)))
 (HASPART (SCOTT (ST X INST (X RUNWAY)
     SURFACE ASPHALT LENGTH (6000 FEET))))

FINDR(USE TELEPHONE XXA))
 (USE (TELEPHONE COMMUNICATION))

FINDR(USE(TELEPHONE PAPERWEIGHT))
 (USE PAPERWEIGHT)

FINDR( ONTOP ((ST X USE(X COMMUNICATION)) TABLE1))
 (ONTOP (TEL1 TABLE1))

FINDR(ONTOP(( ST X USE (X PAPERWEIGHT)) TABLE1))
 (ONTOP (TEL1 TABLE1))

FINDR(ONTOP((ST X USE PAPERWEIGHT) TABLE1))
 (ONTOP (TEL1 TABLE1))

FINDR(ONTOP (TEL1(ST X INST(X TABLE))))
 (ONTOP (TEL1 TABLE1))

FINDR( HASPART( AIRFIELD (ARUNWAY)))
 (HASPART (AIRFIELD (A RUNWAY)))

FINDR(HASPART( EACH AIRFIELD)(A RUNWAY)))
 (HASPART (AIRFIELD (A RUNWAY)))

FINDR(HASPART ( ( ONE AIRFIELD) (A RUNWAY)))
 NIL
```

```
FINDR( HASPART( AIRFIELD (ST X INST (X RUNWAY))))
 (HASPART (AIRFIELD (A RUNWAY)))

FINDR(HASPART(SCOTT (A RUNWAY)))
 (HASPART (SCOTT (ST X INST (X RUNWAY)
      SURFACE ASPHALT LENGTH (6000 FEET))))

FINDR(HASPART (SCOTT (ST X INST(X RUNWAY) SURFACE ASPHALT)))
 (HASPART (SCOTT (ST X INST (X RUNWAY)
      SURFACE ASPHALT LENGTH (6000 FEET))))

FINDR(HASPART(SCOTT (ST X LENGTH(6000 FEET))))
 (HASPART (SCOTT (ST X INST (X RUNWAY)
      SURFACE ASPHALT LENGTH (6000 FEET))))

FINDR(HASPART (SCOTT (ST X LENGTH (5280 FEET))))
     NIL

FINDR(HASPART (SCOTT (ST Z LENGTH(6000 FEET) SURFACE ASPHALT))))
 (HASPART (SCOTT (ST X INST (X RUNWAY)
      SURFACE ASPHALT LENGTH (6000 FEET))))

VFIND( (X) (ONTOP TEL1 Z))
 ((Z . TABLE1))

VFIND( (Q) (ONTOP Q TABLE1))
 ((Q . TEL1))

VFIND( (X) (INST X TELEPHONE))
 ((X . TEL1))

VFIND( (X) ((INST X TELEPHONE)(ONTOP X TABLE1)))
 (FAILFIND (ONTOP X TABLE1) ((X . TEL1)))

VFIND((X)(HASPARTSET ARM(L FINGER X)))
 ((X . 5))

VFIND((X)(HASPARTSET ARM(L FINGER X)))
 ((X . 5))

VFIND( (X)(HASPARTSET HUMAN (L FINGER X)))
 ((X . 10))

VFIND( (X) (INST JOHN X))
 ((X . BOY))

VFIND( (X) (INST X PERSON))
 ((X . JOHN))
```

3. <u>Internal Operation</u>

In normal program operation, only the system response to a question is printed out. By using the LISP function "TRACE," some of the internal operations are shown. The function is applied to VFA1 and gives its arguments and values during the course of answering a question. The question asked below was discussed in Sec. II-D. (DNC means "don't care," UNK means "UNKNOWN," and ALG means "algorithm").

```
VFIND( (X) (INST X PERSON)  )

 ARGS OF VFA1
 (INST X PERSON)
 ((X . UNK))

 ARGS OF VFA1
 (COMPUTES INST ALG)
 ((ALG . UNK))

 VALUE OF VFA1
 ((ALG ST X RULE (FREEVARS (U V W)
     PRED (IMPLIESS (ANDD (INCLUDE V W) (INST U V)) (INST U W)))))

 ARGS OF VFA1
 (INCLUDE V PERSON)
 ((V . DNC) (U . UNK))

 VALUE OF VFA1
 ((V . BOY))

 ARGS OF VFA1
 (INST U V)
 ((V . BOY) (U . UNK))

 VALUE OF VFA1
 ((U . JOHN))

 VALUE OF VFA1
 ((X . JOHN))

 ((X . JOHN))

VFIND( (X)(INST JOHN X))
 ARGS OF VFA1
 (INST JOHN X)
 ((X . UNK))

 VALUE OF VFA1
 ((X . BOY))

 ((X . BOY))
```

Appendix B

ALGORITHMS FOR QA2

Appendix B

ALGORITHMS FOR QA2

Complete listings for program QA2 are available from the authors upon request. Here we include descriptions of some of the key algorithms that are embodied in the QA2 theorem-proving program. The Prenex Algorithm translates input statements and questions into the standard form for the theorem-prover. The Subsumption Algorithm is used whenever possible to eliminate newly generated clauses. The Unification Algorithm is the match procedure at the heart of the resolution method.

1. The Prenex Algorithm

The function (PRENEX E) produces the prenex, conjunctive normal form for the first-order predicate calculus statement E. The form of E may be one of the following:

(1) Literal: E = (P a1 ... an) where P is a predicate symbol, and a1, ..., an represent terms which are the arguments of P.

(2) Conjunction: E = (AND e1 ... en) where the ei can again be any well formed statements. The expression (AND e1 ... en) represents the logical statement $e1 \wedge e2 \wedge \ldots \wedge en$.

(3) Disjunction: E = (OR e1 ... en) which represents the statement $e1 \vee e2 \vee \ldots \vee en$.

(4) Implication: E = (IMP e1 e2) or E = (IF e1 e2) which represent the statement e1 => e2.

(5) Equivalence: E = (EQV e1 e2) or E = (IFF e1 e2) which represent the statement e1 <=> e2.

(6) Negation: E = (NOT e) representing the statement ~e.

(7) Universal Quantification: E = (FA (x1 x2 ... xn) e) where x1, x2, ..., xn are any variables. (FA (x1 ... xn) e) represents the logical statement $(\forall x1)(\forall x2) \ldots (\forall xn)e$.

(8) Existential Quantification: E = (EX (x1 x2 ... xn) e) represents the statement $(\exists x1)(\exists x2) \ldots (\exists xn)e$.

Any symbol which occurs in a literal and which has not been quantified is assumed to denote a constant--e.g., in the statement:

(FA (X) (IF (IN X BOY)(IN X PERSON)))

BOY and PERSON are treated as constants.

The prenex conjunctive normal form of a statement E is a list of clauses (C1 ... Cn) where each Ci is a list of literals (L1 ... Lm).

First PRENEX calls PRENEX1 with the argument E. PRENEX1 performs the following tasks:

(1) If E = (NOT e) then (PRENEX1 e) is called. A flip-flop switch is set to indicate that e is negated. This has the effect of taking negation symbols inside the expression.

(2) If E = (AND e1 ... en) or E = (OR e1 ... en) then (PRENEX1 ei) is called for each i, i = 1, ..., n. The resulting clause lists are combined into a single OR or AND list of the results. The negation flip-flop is used to avoid multiple negations.

(3) If E = (IF e1 e2), (IMP e1 e2), (IFF e1 e2), or (EQV e1 e2) then the connectives are transformed into connectives using AND, OR, and NOT, and PRENEX1 of the transformed expression is called--e.g., (IF e1 e2) is transformed into (OR (NOT e1) e2).

(4) If E is a quantified expression:

  (a) If the essential quantifier is universal (FA if the negation flip-flop is not set and EX if the switch is set) the variables x1, ... xn are added to a free variable list and (PRENEX1 e) is called.

  (b) If the essential quantifier is existential, each of the variables x1, ..., xn is bound to a generated Skolem function whose arguments are all of the variables on the free variable list at this point, and (PRENEX1 e) is called.

(5) If E = (P a1 ... an) is an atomic formula, then:

  (a) All of the variables in E which occur on the list of bindings are replaced by their corresponding Skolem functions.

(b) The expression (((P a1 ... an))) or
((NOT (P a1 ... an))) is returned as the value of PRENEX1 of E (according to the flip-flop), and is the conjunctive normal form of E = (P a1 ... an).

When PRENEX1 exits with the list of clauses corresponding to the original statement E, PRENEX performs a tautology and subsumption check on the clauses so as to return a list of non-tautologic, inequivalent clauses, each of which is of minimal length.

2. The Subsumption Algorithm

The function (SUBSUME C1 C2) determines whether or not the clause C1 subsumes the clause C2. That is, it sees whether or not there is a substitution θ such that

$$(C1)_\theta \subset C2 \quad .$$

The following algorithm is used:

Let A1 = (L1 ... Ln), B1 = (M1 ... Mm) be the literals of C1 and C2 respectively. Two association lists AL1 and AL are used for keeping track of the various substitutions computed during the subsumption test. Initially AL1 and AL are both set to NIL.

The function (SUBSM1 A1 B1) is called to see if the current literal list A1 subsumes the list B1 with respect to the substitution on AL.

(1) If A1 is ever NIL the test is successful and SUBSM1 returns true.

(2) If B1 is ever NIL (A1 ≠ NIL) then SUBSM1 returns false.

(3) If neither A1 nor B1 is NIL, the SUBSM2 is called to see if the first literal of A1 subsumes the first literal of B1, when AL1 is first set to AL. If the test is successful then the substitution list AL1 is modified to include the match and then a test is made (recursively) with SUBSM1 called with A1 = (L2 ... Ln), B1 = (M1 ... Mm), and AL = AL1. If this is successful, then SUBSM1 returns truth. If this test is not successful, or if L1 did not subsume M1, then SUBSM1 is called with A1 = (L1 ... Ln), B1 = (M2 ... Mm), and AL restored to its previous value.

The algorithm has the effect of first trying to map L1 into C2, then $(L2)\theta_1$ into C2 (where θ_1 is the substitution that was needed to reduce L1 to a literal of C2) and so on. If at any point $(Li)\theta_{i-1}$ cannot be mapped into C2, the process backs up one step and retries on L_{i-1}. It terminates when all of the Li are mapped into C2 or when L1 has been tried on all of the literals of C2.

3. The Unification Algorithm

The function (RESOLVE L1 L2 C1 C2) produces the resolvent of clauses C1 and C2 on literals L1 of C1 and L2 of C2.

For a resolvent to exist it is necessary that L1 and L2 be of the form of complementary literals--e.g.,

L1 = (P a1 ... an), L2 = (NOT (P b1 ... bn))

and that

(P a1 ... an), (P b1 ... bn) be unifiable.

The function (MATCH A B) determines whether or not the lists of arguments A = (a1 ... an), B = (b1 ... bn) are unifiable, and if so, produces the most general unifier θ.

Let AL1 and AL2 be two association lists, each initially empty, which will hold the bindings of variables on A and B, respectively. The function (MATCH A B) performs the following operations:

(1) The function VAL applied to A1 and AL1 causes (a) any variables in A1 not already on AL1 to be added to AL1 and bound to distinct new variables, and (b) the term a1′ to be generated by replacing each variable in a1 by its value from the AL1 list. VAL is then applied to b1 and AL2, except that new variables in b1 are initially bound to themselves.

(2) The function MATCH1 is then applied to a1′ and b1′, as follows. If a1′ is a variable and b1′ is not, and a1′ does not occur in b1′, then b1′ is substituted for every occurrence of a1′ in the value of every variable on AL1 and AL2. If b1′ is a variable, then a1′ is substituted for b1′ in the values on AL1 and AL2. Otherwise MATCH1 reports failure.

If both a1′ and b1′ are functional terms, such as a1′ = (f s1 ... sr), b1′ = (g t1 ... tk), then we must have f = g and the function MATCH1 must succeed, recursively, with each pair of corresponding arguments from (s1 ... sr) and (t1 ... tk). Otherwise MATCH1 returns NIL.

(3) If MATCH1 succeeds in a1′ and b1′, then MATCH is called on the lists (a2 ... an), (b2 ... bn). If both lists are empty, the unification was successful and the substitutions on AL1 and AL2 represent the most general unifier θ.

REFERENCES

1. R. F. Simmons, "Answering English Questions by Computer: A Survey," COMM. ACM, Vol. 8, No. 1 (January 1965).

2. K. M. Colby and H. Enea, "Heuristic Methods for Computer Understanding of Natural Language in Context-Restricted On-Line Dialogues," Dept. of Computer Sciences, Stanford University (1966).

3. J. McCarthy, "Situations, Actions, and Causal Laws," Memo No. 2, Stanford Artificial Intelligence Project, Stanford University (July 1963).

4. R. Quillian, AFIPS Proc. SJCC, Vol. 30 (1967).

5. R. F. Simmons, "An Approach Toward Answering English Questions From Text," AFIPS Proc. FJCC, Vol. 29 (1966).

6. J. R. Slagle, "Experiments With a Deductive Q-A Program," COMM ACM Vol. 8, No. 12 (December 1965).

7. F. B. Thompson, "English for Computer," AFIPS Proc. FJCC, Vol. 29 (1966).

8. J. A. Craig et al., "DEACON: Direct English Access and Control," AFIPS Proc. FJCC, Vol. 29 (1966).

9. J. W. Weizenbaum, "ELIZA - A Computer Program for the Study of Natural Language Communication Between Man and Machine," COMM. ACM, Vol. 9, No. 1 (January 1966).

10. B. Raphael, "A Computer Program Which Understands," AFIPS Proc. FJCC, Vol. 26 (1964).

11. J. McCarthy, "Programs with Common Sense," Memo No. 7, Stanford Artificial Intelligence Project, Stanford University (September 1963).

12. B. Raphael, "SIR: A Computer Program for Semantic Information Retrieval," MAC-TR2, Project MAC, MIT (June 1964).

13. J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," J. ACM, Vol. 12, No. 1 (January 1965).

14. L. Wos, et al., "The Unit Preference Strategy in Theorem Proving," AFIPS Proc. FJCC, Vol. 26 (1964).

15. F. Black, A Deductive Question-Answering System, Harvard University Ph.D. Thesis (1964).

16. D. C. Cooper, "Theorem Proving in Computers," in Advances in Programming and Non-Numerical Computation, L. Fox (ed.), (Pergamon Press, 1966).

17. E. Mendelson, Introduction to Mathematical Logic (van Nostrand, 1964).

18. T. P. Hart, "A Useful Algebraic Property of Robinson's Unification Algorithm," Memo No. 91, AI Project, Project MAC, MIT (1965).

19. J. R. Slagle, "Automatic Theorem-Proving with Renameable and Semantic Resolution," J. ACM (in press).

20. Kalish and Montague, Logic: Techniques of Formal Reasoning (Harcourt, Brace and World, 1964).

21. B. Raphael, "Aspects and Applications of Symbol Manipulation," Proc. 1966 National Conference, ACM (1966).

22. J. A. Robinson, American Mathematical Society Symposia on Applied Mathematics, XIX (1967), "A Review of Automatic Theorem-Proving," Rice University (to be published).

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Stanford Research Institute 333 Ravenswood Avenue Menlo Park, California 94025	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP N/A

3. REPORT TITLE

RESEARCH ON INTELLIGENT QUESTION-ANSWERING SYSTEMS

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Scientific Report Interim

5. AUTHOR(S) (Last name, first name, initial)

Green, Claude Cordell Raphael, Bertram

6. REPORT DATE May 1967	7a. TOTAL NO. OF PAGES 53	7b. NO. OF REFS 22
8a. CONTRACT OR GRANT NO. AF 19(628)-5919 b. PROJECT, Task, Work Unit Nos. 4641- 02 46410201 c. DOD ELEMENT 62405454 d. DOD SUBELEMENT 674641	9a. ORIGINATOR'S REPORT NUMBER(S) SRI Project 6001 Scientific Report 1 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) AFCRL-67-0370	

10. AVAILABILITY/LIMITATION NOTICES

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Air Force Cambridge Research Laboratories (CRB) L. G. Hanscom Field Bedford, Massachusetts 01730

13. ABSTRACT

This report describes progress toward an "intelligent question-answering system"--a system that can accept facts, retrieve items from memory, and perform logical deductions necessary to answer questions. Two versions of such a system have been implemented, and the authors expect these to be the first in an evolving series of question-answerers.

The first system, QA1, is based upon relational information organized in a list-structured memory. The data consist of general facts about relations as well as specific facts about objects. QA1 has limited deductive ability.

QA2 is based upon formal theorem-proving techniques. Facts are represented by statements in the predicate calculus. Although the memory organization is simpler than that of QA1, the sophisticated logical abilities of QA2 result in greater question-answering power.

The report gives examples of the performance of QA1 and QA2 on typical problems that have been done by previous question-answerers, and describes plans for extending the capabilities of QA2.

DD FORM 1 JAN 64 1473

UNCLASSIFIED

Security Classification

UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Question-Answering Systems						
Theorem-Proving Programs						
Memory Organization						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year, or month, year. If more than one date appears on the report, use date of publication.

7a. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8a. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through ______________."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through ______________."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through ______________."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, roles, and weights is optional.

UNCLASSIFIED

Security Classification